Onderzoeksrapport Nr. 7615

A BRANCH-AND-BOUND PROCEDURE FOR THE SINGLE-MODEL DETERMINISTIC ASSEMBLY LINE BALANCING PROBLEM

by

Frans Van Assche[1] en Willy S. Herroelen[2]

Wettelijk Depot : D/1976/2376/27

[1]Department of Applied Economic Sciences, Katholieke Universiteit Leuven

[2]Associate Professor, Department of Applied Economic Sciences, Katholieke Universiteit Leuven.

The authors are greatly indebted to Prof. Y.M.I. Dirickx for his valuable suggestions and constructive criticism.

ABSTRACT.

An algorithm is described which permits the computation of optimal solutions for the single-model deterministic assembly line balancing problem. The procedure is a branch-and-bound algorithm equipped with dominance rules, bounding arguments and reliable branching heuristics. Computational results are given, along with an example of the type problems solved. Computational results indicate the method to be more than competitive to ones previously reported.

A BRANCH-AND-BOUND PROCEDURE FOR THE SINGLE-MODEL DETERMINISTIC ASSEMBLY LINE BALANCING PROBLEM.

The well-known single-model deterministic assembly line balancing problem involves the assignment of work elements of a fixed duration to a minimum number of work stations along an assembly line, without exceeding the cycle time for each station and satisfying the precedence relations between the elements [7]. A number of exact and heuristic methods have been proposed for the solution of the problem (for a review see [1], [2], [3], [7], [9]). It would appear that among the exact methods, the 0-1 integer programming approach of Thangavelu and Shetty [14] is the most efficient as the authors claim it to be approximately 50 percent faster than the dynamic programming approach of Held, Karp and Shareshian [6] on the same range of problem sizes. The purpose of this study is to develop a branch-and-bound method incorporating bounding arguments, dominance rules and reliable search heuristics, which will guarantee the optimality of the solutions obtained.

THE BRANCH-AND-BOUND METHOD.

Essentially the branch-and-bound method presented is a tree search procedure. Each iteration of this procedure begins with a node representing the assignment of work elements to a certain work station. The problem associated with the node - the assignment of the remaining work elements to the remaining work stations - is inspected to see if it will yield an immediate solution.

If it is determined that no immediate solution can be found, the procedure branches into a number of descendant nodes corresponding to the next station assignments. For each such node a lower bound is computed. The program than takes a global look at all nodes from which no branching has yet taken place, and chooses the node with the smallest lower bound for the next iteration. If an immediate solution is found for any node in the tree, then a solution is guaranteed for all its ancestors.

The gist of the approach discussed below is to try to limit the explicitly-enumerated nodes in the search tree by feasibility, dominance and bounding arguments so that it is not necessary to search for an optimal station assignment over all possible assignments, but only over a specified subset.

Generation of work station assignments.

Work element U_i is a predecessor of work element U_j if the assembly process requires work element U_i to be completed before U_j can be begun. If U_i is a predecessor of U_j, then U_j is said to be a successor of U_i. U_i is an immediate predecessor of U_j if (1) U_i is a predecessor of U_j, and (2) no successor of U_i is a predecessor of U_j. If U_i is an immediate predecessor of U_j, then U_j is an immediate successor of U_i. Thus in the precedence diagram of Figure 1, work element U4 has four predecessors (U1, U6, U7, U8), two successors (U5, U11), three immediate predecessors (U6, U7, U8), and one immediate successor (U5).

A work element is eligible if (1) it has not yet been assigned to a work station, and (2) all of its immediate predecessors have been already assigned.

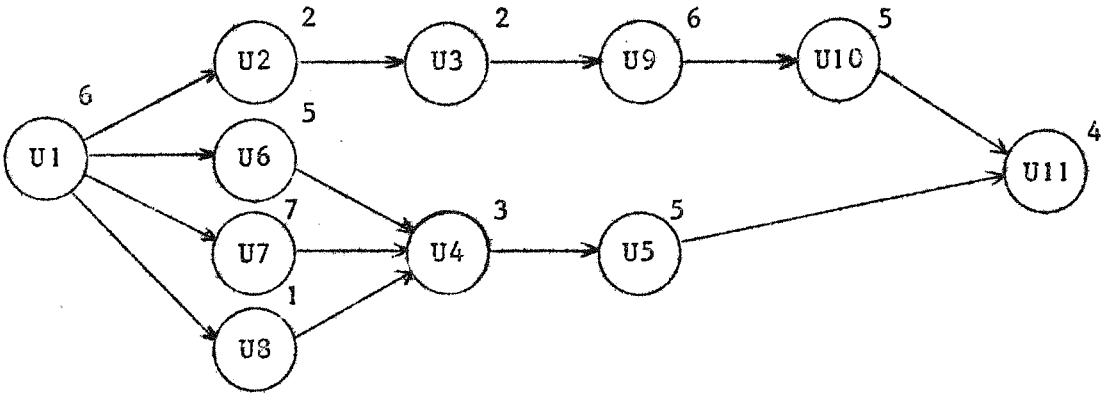

Figure 1 : Precedence diagram of Jackson's 11-element problem [8]. Symbols inside a circle represent work elements, numbers outside a circle represent element durations.

We begin the iteration with an empty work station, and construct an order-free list of all eligible work elements (the PEND-list). We move down the list until a work element U_1 is found which when added to the work station will not cause the station time to exceed the cycle time. U_1 is assigned to the work station and a pointer is moved to the position of U_1 in PEND. If the station time is smaller than the cycle time, all of the work elements made eligible by the assignment of U_1 to the work station are now inserted in PEND in a order-free manner. We continue down PEND adding new elements U_2, U_3, ..., U_j to the work station whenever possible until we reach the end of PEND (the pointer is then set to the position of U_j), with the restriction that elements made eligible by the assignment of element U_j to the work station are not inserted in PEND if the assignment of U_j caused the station time to equal the cycle time. The set of elements in the work station will form a new node in the branch-and-bound tree provided that (1) this work station is not a proper subset of some other work station generated during this iteration, and (2) it is not fathomed by the dominance tests explained below.

We then (1) remove from the work station element U_j and delete its immediate followers in PEND, (2) continue on that portion of PEND after U_j, looking for new work stations (if no new station assignments can be found, the pointer in PEND is reset to U_{j-1} and the successors of U_{j-1} are deleted from PEND). The generation stops when the work station is empty and no element in PEND after the pointer position can be assigned.

It should be noted that this generation procedure is an updated version of the "ready list" concept presented by Nevins [12], except that (1) our PEND-list is not arranged in order of decreasing operation times with the effect that immediate followers are simply inserted at the end of PEND and deletion of immediate followers can be done by simple truncation, which yields considerable savings in computation time, and (2) the procedure presented here finds all feasible work stations which are then subjected to dominance tests.

Elimination of feasible work station assignments.

It is clear that the procedure for generating feasible work station assignments as described above has the major drawback that the number of feasible work station assignments can be very large for even small problems, depending upon the duration of the work element times, the cycle time and the complexity of the precedence diagram. An obvious means of improving the procedure is to apply techniques for reducing the total number of feasible work station assignments generated, while still guaranteeing the production of an optimal solution.

Apart from the rule - inherent to the above generating procedure - that only work stations (i.e., nodes in the search tree) that are not a proper subset of some other work station generated during the same iteration are explicitly generated, we follow Jackson [8] and Reeve [13] by including the rule that a node corresponding to a sequence (i.e., work station assignments along the path of the search tree leading to the particular node) is crossed off if it is a subset of a sequence corresponding to another node (still not crossed off) at the same level of the search tree.

It should be noted that our procedure could also incorporate Jackson's dominance rules III and III' in the following manner :

Rule III : Successively cross off nodes X for which there is another node Y (still not crossed off), such that : (a) there is just one work element x in X which is not also in Y, and (b) there is some work element y in Y, which is not in X, which has performance time at least as great as that of x, and such that arrows can be followed from y to any operation z for which there is an arrow from x to z.

Rule III' : Successively cross off nodes X for which there is another node Y (still not crossed off), such that there is some work element y in Y but not in X, which has performance time at least equal to the sum of the required

times for operations X-Y, and such that arrows can be followed from y to any operation not in X to which arrows can be followed from operations in X-Y (where X-Y is the set of work elements in X but not in Y).

Preliminary tests with our algorithm forced us to conclude, however, that the payoff of including these rules was not worth the cost caused by the complication of using them and the resulting increase in computation time, especially for precedence networks where the technological restrictions are rather strong (see also Jackson [8] and a theorem by Kohler and Steiglitz [10] which states that the computational requirements for the general branch-and-bound algorithm may increase when a stronger dominance relation is used).

Testing for an immediate solution.

Let

T^* = sum of performance times of all work elements (total work content)

T = sum of operation times of work elements which so far have been assigned

c = cycle time

A node in the search tree is a candidate for an immediate solution if $\left\lceil \frac{T^* - T}{c} \right\rceil = 2$, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. If such a node sprouts a descendant node for which the station time t_s is such that $T^* - T - t_s \leq c$, an immediate solution is obtained by assigning the remaining elements to one single work station.

The lower bound.

An obvious lower bound for each node of the search tree is obtained as

$$LB = N + \left\lceil \frac{T^* - T}{c} \right\rceil ,$$

where N denotes the number of work stations which so far have been used, i.e. the corresponding level of the search tree.

As Nevins [12] pointed out already, the major drawback of this bound is its relative insensitivity in determining the node from which branching should continue. For this reason a node of the search tree is also assigned a penalty. The procedure then branches from the node with the smallest lower bound, where ties are resolved by branching from the node with the smallest penalty. This penalty measure is obtained as follows.

In their algorithm for the multiple-resource constrained project scheduling problem, Davis and Heidorn [4] use in their resource-based elimination criterion an expression which, when translated to line balancing terms, reads

$$T = \max \ (0, T^* - (N^* - N)c) \ ,$$

where N^* denotes an appropriate upper bound on the number of stations. Applying this elimination criterion to the line balancing problem, would eliminate all nodes for which $T^* - T > (N^* - N)\ c$ since it would be impossible then to assign the remaining work elements to the remaining $N^* - N$ work stations without exceedi the cycle time for at least one station. It should be noted, however, that the incorporation of such a dominance rule in our procedure is impossible, since we do not bother to compute an upper bound on the number of work stations needed. Instead we compute for each node in the search tree a penalty

$$P = \frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil} - \alpha + \beta \ ,$$

where the information content of $\left\lceil \frac{T^* - T}{c} \right\rceil$ can be regarded as equivalent to $N^* - N$, i.e., the number of work stations remaining to be assigned, and α and β are small-valued parameters defined below.

This penalty measure sprouts from the following philosophy. The first part $\frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil}$ - rooted in the dominance rule discussed above and largely equivalent to the score computed by Nevins [12] - represents the average time which must be assigned to the remaining work stations. In breaking the lower bound-ties

by choosing the node with the smallest P-value - where P is largely determined by $\frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil}$ - one so forces the idle time towards stations generated in the bottom part of the search tree.

The parameter α may be computed as

$$\alpha = \frac{k}{\left\lceil \frac{T^* - T}{c} \right\rceil} ,$$

where k = a constant given to the program by the problem solver. Alternatively, one may wish to let the program itself compute an appropriate value for α without the intervention of the problem solver. In the computer experiment described below, k was arbitrarily defined as

$$k = \max \left\{ 1 ; \frac{N^*_c - T^*}{\sqrt{c}} \right\}$$

The effect of using such a parameter in the penalty measure is that the higher the value of α, the greater would be the decrease in the penalty associated with nodes at lower levels of the search tree. In doing so, we hope that the procedure would not "lose" too much of its time in backtracking (within the same lower bound and the same or nearly the same $\frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil}$ - value) to nodes at level n-1 and higher before branching from nodes at level n. Stated another way, the use of α favours a depth first search of our branch-and-bound procedure.

In order to break ties between nodes with the same value for the lower bound, $\frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil}$ and α, the parameter β is introduced. The philosophy for doing so is

somewhat more involved. We compute β as

$$\beta = \frac{a}{b} \quad ,$$

where a proper choice of a and b allows the following four strategies :

1. a is positive; b = number of work elements associated with the node (NC). With this strategy, the larger the number of work elements assigned to the station - say, node - for which the penalty is computed, the lower the penalty P.

2. a is negative; b = NC. If the nominator a takes on a negative value, the smaller NC the smaller the penalty P. This strategy resembles the "largest time rule" (Tonge [17]) in which work elements with the largest operation time are chosen, since for a fixed cycle time c, the smaller NC the higher the duration of the tasks involved.

3. a is positive; b = number of work elements ready for assignment. b is computed as NP-NC, where NP equals the number of work elements in the PEND-list discussed above. With this strategy the higher b, the lower the penalty P. This strategy follows the philosophy of the "most immediate followers" heuristic (Tonge [17]).

4. a is negative; b = NP-NC. With this strategy the larger the number of work elements "ready" (eligible), the higher the value of P.

Switches from strategy 1 to 2 and from strategy 3 to 4 are made possible by making our program to compute the a-value as

$$a = \eta \left\{ 1 - \frac{[(N^* - N)c - (T^* - T) + \varepsilon]}{c\theta} \right\} \quad ,$$

where

η = small positive value (e.g. 0.1) to assure that β does not interfere with α and $(T^* - T) \Big/ \left| \frac{T^* - T}{c} \right|$;

ε = small positive value to assure that the nominator - i.e., the expected idle time for the remaining number of work stations - does never equal $c\theta$, which would have for effect that $a = 0$ and consequently $\beta = 0$;

θ = a constant between 0 and 1 set by the problem solver.

The overall effect of computing the a-value as such, is that the sign of a can be changed by the solution procedure as it proceeds through the search tree. If for example $\theta = 0.1$, a becomes positive (negative) if the expected idle time for the remaining work stations falls below (climbs above) 10 percent of the cycle time, producing a switch between the strategies as discussed above.

In general the global effect of β at a node on a certain level of the search tree, is to increase the number of sprouted nodes to be examined at the lower level, thus increasing the probability that an optimal solution is found along the path emanating from the node with the lowest P-value, but with the disadvantage that more nodes have to be examined (this effect is reversed by a change in sign of a).

Experimentation with the program on a set of test problems, as discussed below, revealed that good results were obtained if

$$\text{effect of } \frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil} > \text{effect of } \alpha > \text{effect of } \beta.$$

The rationale for these reliable branching heuristics being discussed - reliable, because an optimal solution is always guaranteed - we are now in a position to present the overall structure of the solution algorithm.

The algorithm

The branch-and-bound procedure for the single-model deterministic line balancing problem comprises the following steps.

Preparation : Read the element numbers, element durations, immediate successors, the cycle time c, and the value of the parameters η, θ, ε.

Step 0 (Initialize). Generate the node $\emptyset$, corresponding to an empty work station. Compute the initial lower bound on the number of work stations needed as

$$N^* = \left\lceil \frac{T^*}{c} \right\rceil$$

Compute the value of $k = \max \left\{ 1; \frac{N^*c - T^*}{\sqrt{c}} \right\}$

Step 1. Set N=1, the level of the search tree.

Step 2.a. Generate corresponding feasible N-level station assignments (nodes).

2.b. Test for an immediate solution. If an immediate solution is obtained, go to step 2.c.; otherwise go to step 3.

2.c. An optimal solution to the problem is obtained by assigning the remaining work elements to a single station. Stop.

Step 3. Fathom all nodes which do not satisfy the dominance test.

Step 4. For each node, compute the corresponding lower bound

$$LB = N + \left\lceil \frac{T^* - T}{c} \right\rceil \quad \text{and penalty} \quad P = \frac{T^* - T}{\left\lceil \frac{T^* - T}{c} \right\rceil} - \alpha + \beta.$$

Step 5. Place all nodes with the same lower bound in a PENALTY-list in order of decreasing P-value.

Step 6. If the PENALTY-list is empty, the previous lower bound on the number of work stations is unattainable. Go to step 7; otherwise, go to step 8.

Step 7. Increase the lower bound on the number of stations, $N^* \leftarrow N^* + 1$. Insert all nodes with $LB = N^*$ (i.e., all nodes not yet on PENALTY) on the PENALTY-list in order of decreasing P-value.

Step 8. Branch to the node placed in the last position of PENALTY; i.e. the node with the current lowest bound, lowest penalty.

Step 9. Update the level of the search tree to be considered; i.e. N = (level of node picked in step 8) + 1. Go to step 2.

A flow chart of the procedure is given in Figure 2.

AN ILLUSTRATIVE EXAMPLE.

Figure 3 illustrates the way in which the branch-and-bound procedure obtained a 5-station balance to Jakcson's problem of Figure 1 for a cycle time of $c = 10$, $k = 0.25$ and $\beta = 0$.

In Figure 3, the circles represent the nodes of the search tree. Inside a circle are indicated the corresponding node number together with the elements assigned to the corresponding work station. Next to each node, we give the corresponding lower bound LB and the penalty P.

Initializing the procedure we generate the empty work station $\emptyset$. As $T^* = 46$ and $c = 10$ the initial lower bound on the number of work stations needed is $\lceil T^* / c \rceil = 5$. Generating the feasible undominated first station assignments, we obtain the two descendant nodes corresponding to {U1, U2, U8} and {U1, U2, U3}. Since both nodes have the same lower bound equal to 5, they are entered in the PENALTY-list in the order (1,2) as node number 2 has the smallest penalty associated with it. In branching from node 2, we generate nodes 3, 4 and 5 corresponding to the feasible second station assignments {U6, U8} , {U7, U8} and {U8, U9} . It should be noted here that the incorporation of Jackson's dominance rule III in our procedure would cross off assignment {U6, U8} as it would be dominated by assignment {U7, U8}. Indeed, the processing time of element U7 is larger than that of element U6 and arrows can be followed from U7 to U4 which is reached by one arrow from U6.

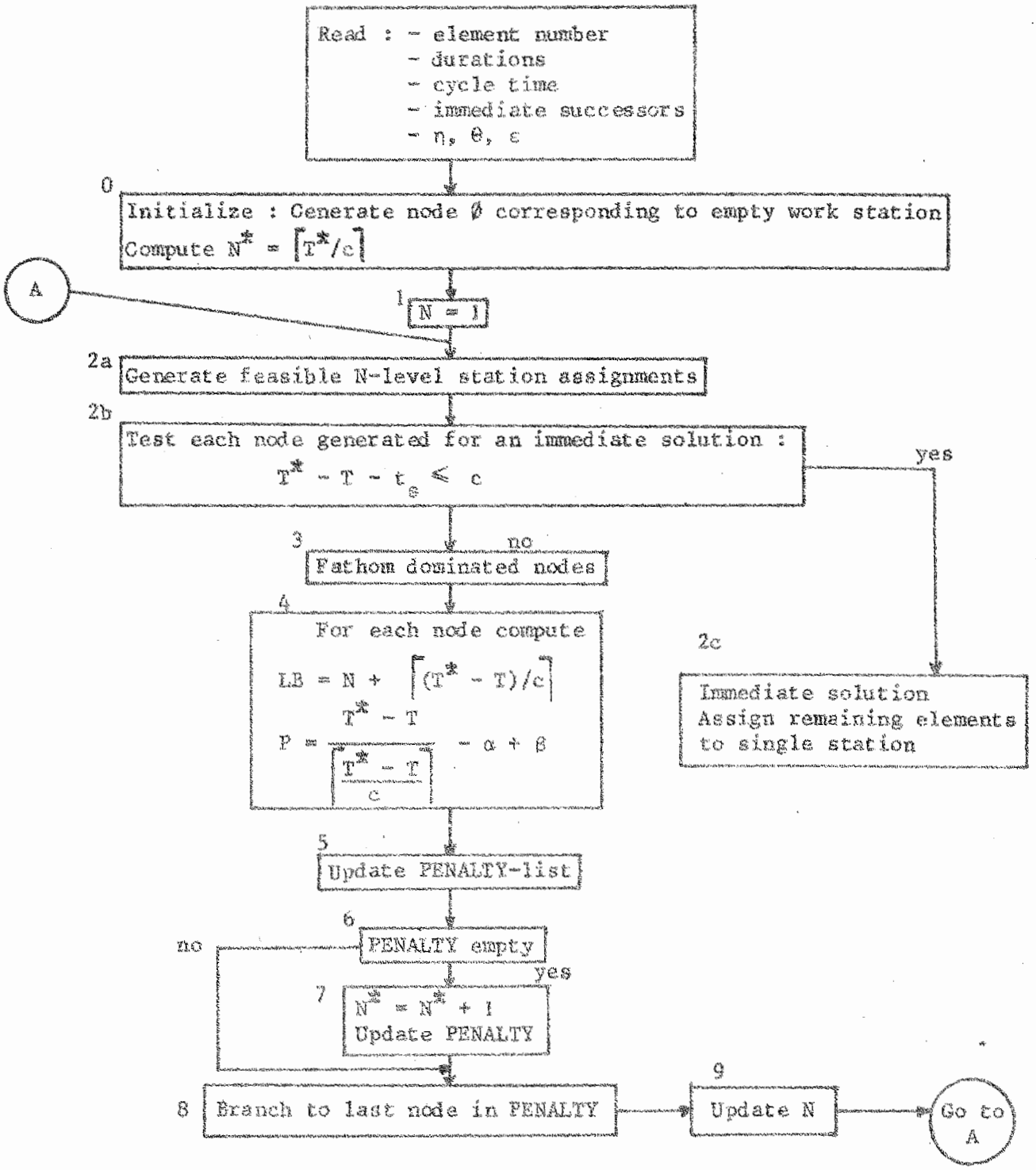

Figure 2. Flow chart of the branch-and-bound algorithm.

Our procedure inserts nodes 3, 4, and 5 in the PENALTY-list in the appropriate order, such that PENALTY now takes the form (3,5,4,1).

Branching continues from node 1, placed last in PENALTY since it has the smallest penalty associated with it. The nodes generated correspond to the second station assignments {U3, U6} , {U3, U7} , {U3, U9} , but are all crossed off since they correspond to assignment sequences {U1, U2, U8, U3, U6} , {U1, U2, U8, U3, U7} and {U1, U2, U8, U3, U9} which are subsets of the sequences already obtained through the generation of nodes 3, 4 and 5. Node 1 is deleted from the PENALTY-list and branching continues from the last node on this list, i.e., node number 4. Node 6 and 7, both generated during this step, are inserted in PENALTY, which now takes the form (3,6,5). Since node 7 has a lower bound of 6 it is not entered on the PENALTY-list. Continuing the branching from node 5 - within the nodes with the same lowest bound, the one with the smallest penalty - the procedure generates node 8, corresponding to the third station assignment {U6, U10} . The updated PENALTY-list now reads (6,8) so that branching continues from node 8 for which $\lceil (T^* - T)/c \rceil = 2$. Node 8 sprouts the only undominated fourth station assignment {U7, U4} for which $T^* - T = 46-37 = 9 < c$, indicating that an immediate solution is found. The remaining elements {U5, U11} are assigned to one single station and the procedure stops.

COMPUTATIONAL RESULTS

The branch-and-bound procedure has been programmed in FORTRAN IV for the IBM 370/158 computer. Computational results were obtained for eight problem sets. The first problem is an 11-element problem described by Elmaghraby [5] solved for c=12. Problem set 2 consists of Jackson's 11-element problem [8] solved for c=10 and c=12, while the third is a 19-element problem described by Moodie [11] solved for c=179. The fourth and eight problem sets consist of Tonge's 21- and 70-element problem [16] solved for various cycle times.

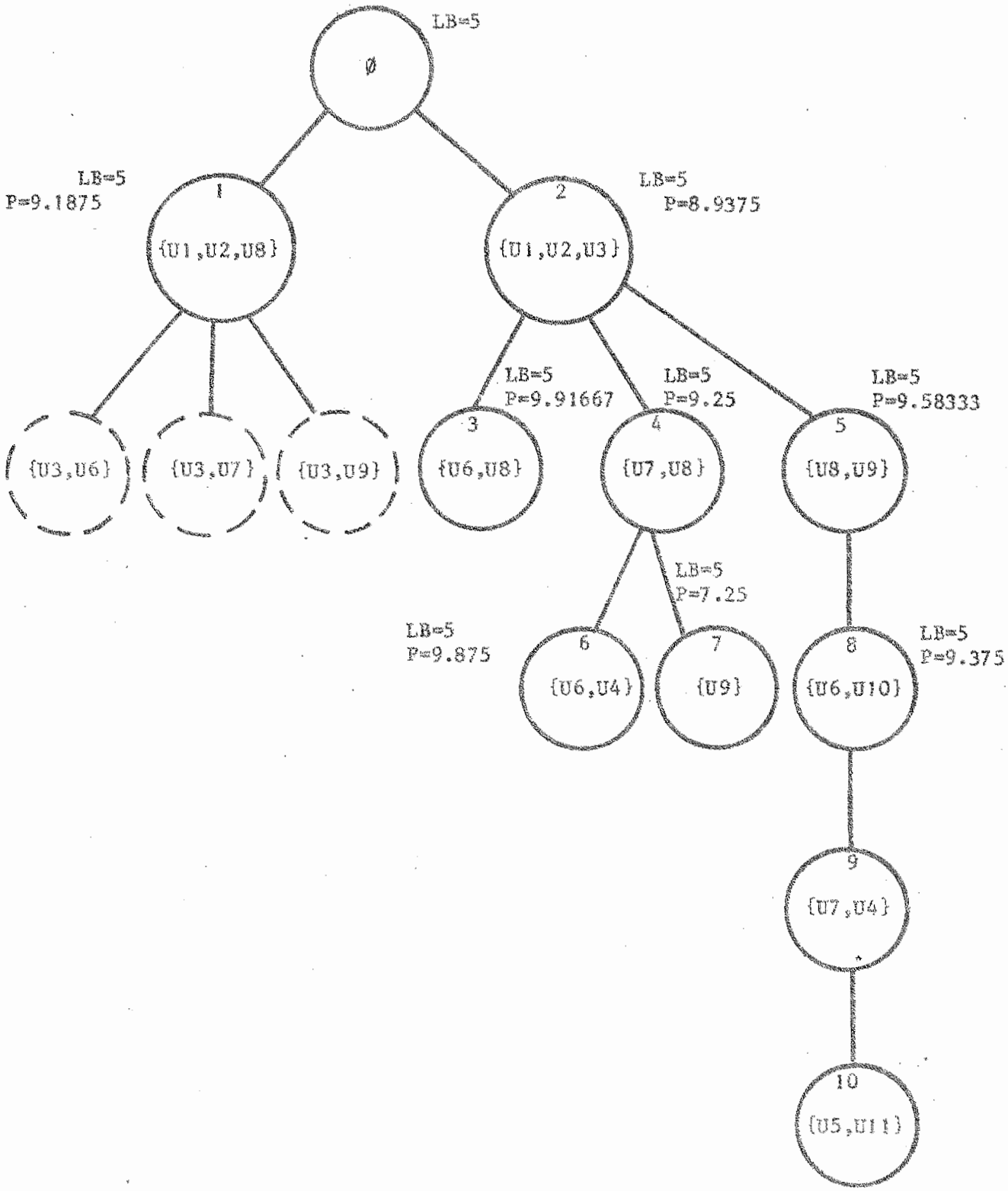

Figure 3. Branch-and-bound tree for the illustrative problem example.

Problem set 5 is a 42-element problem borrowed from Reeve [13] and problem 6 is the well-known Kilbridge & Wester 45-element problem [9] with c=69. Problem 7, at last, is Moodie's 48-element problem [11] solved for c=161. For those problems, where certain work element times t exceed the cycle time c, the computer program breaks the line and sets up q work stations in parallel where q is the smallest integer for which $qc \geq t$. A set S of work elements is then created, and assigned to each of these parallel stations. Members of S are the work element whose time exceeds c as well as other elements provided that the ordering restrictions are satisfied and the sum of work element times in S does not exceed qc. Since $qc-t < c$, this implies that there is only one element in S to exceed the cycle time (see also Tonge [16] and Nevins [12]).

Each of these problems was solved with $k = \max \left\{ 1; \frac{N^*c - T^*}{\sqrt{c}} \right\}$

and seven different values of the β-parameter computed as

$$\beta_1 = 0,\ \beta_2 = + \frac{0.1}{NC}\ ,\ \beta_3 = - \frac{0.1}{NC}\ ,\ \beta_4 = + \frac{0.1}{NP-NC}\ ,$$

$$\beta_5 = - \frac{0.1}{NP-NC}\ ,\ \beta_6 = + \frac{0.1\ XI}{NC}\ ,\ \text{and}\ \beta_7 = + \frac{0.1\ XI}{NP - NC}\ ,$$

where $XI = \{1 - [(N^* - N)c - (T^* - T) + \varepsilon] / c\theta\}$ and $\theta = 0.1$, $\varepsilon = 0.000001$.

The computational results using the branch-and-bound algorithm are depicted in Table I. Under the heading "Problem characteristics" this table gives for each of the solved problems the number of work elements, the cycle time, the optimal number of work stations as reported in the literature, and the number of precedence relations. For each problem we also give the number of constraints and the number of variables needed in the formulation of the ILP-model of Thangavelu & Shetty [14]. The number of constraints is determined as M (an upper bound on the number of stations) + N (number of work elements) + R (number of precedence relations) + 1. The number of 0-1 variables was obtained as (number of elements) x (number of stations).

Under the heading "Branch-and-bound Solution" we depict for our algorithm the number of nodes generated, the β-parameter that resulted in the lowest CPU-time (excluding input and output) needed for obtaining the optimal solution, and the number of stations (normal and parallel) generated. We also indicate the CPU-time in seconds obtained by applying the algorithm equipped with the β-parameter indicated, together with the variance of CPU-times needed by our method when employing the remaining six β-parameters. Finally, we list the mean and variance of the number of nodes generated, also for the remaining six β-parameters. Mean and variances are not indicated for those problems for which the use of one or more of the β-parameters did not yield an optimal solution within 3 minutes of CPU-time and/or after the generation of 3500 nodes.
Under the heading "Other algorithms" we list the best results obtained by other algorithms reported in the literature, i.e., the number of stations generated, and if the algorithm was computerized, the CPU-time (in seconds) together with the computer used.

It can be seen from Table I that the branch-and-bound algorithm produced an optimal solution for all problems tested within very encouraging CPU-times. For the set of problems also solved by Thangavelu & Shetty, direct comparison of CPU-times is very difficult since different computers are used. Taking for given that the IBM 370/158 is approximately five times faster than the UNIVAC 1108, our branch-and-bound algorithm is faster than the ILP-approach except for two of the 21-element problems (c = 14,20) for which the CPU-times obtained, however, are still very competitive.

For the large realistic 70-element problems, direct comparison with other optimal algorithms is impossible since, to the best of our knowledge, this problem set so far resisted a solution with optimal procedures. The best heuristic results (measured by the number of stations generated) obtained for this set of problems have been reported by Nevins [12], who, unfortunately,

does not indicate any CPU-times. Our branch-and-bound algorithm, using the β-parameters indicated in Table I, solved these 14 problems in an average CPU-time of 37.471 seconds with a standard deviation of 39.15. Especially encouraging are the results obtained for the 70-element problems with cycle times 83, 86, 89 and 95, four of the most difficult problems to solve among this set (see also Nevins [12] and Tonge [17]).

As concerns the effect of the β-parameters on the CPU-time required and the number of nodes generated by the branch-and-bound algorithm, the results obtained reveal this effect to be rather modest for the small and intermediate sized problems. For the larger 70-element problems the β-parameters start playing a more relevant role. In effect, some of the problems among this set could not be run to completion with certain of the β-parameters due to excessive CPU-time needed ($>$ 3 minutes) or the excessive number of nodes generated ($>$ 3500). Detailed results for this 70-element problem are given in Table II.

It is clear from this table that none of the β-parameters used, consistently yields the best results. Where a problem, eventually could not be solved using $\beta_1 = 0$, the use of β_5 or β_7 did always help. This result is not so surprising. For the problems tested, the use of β_5 resulted in the fact that relatively "few" nodes (having the same lower bound) were explicitly kept track of at each stage of the search tree. This yielded a favourable effect on the solution efficiency for those problems which did require a large number of nodes to be generated in order to obtain a solution. On the same set of problems, parameter β_7 allowed the search to proceed towards the lower levels of the search tree, before a switch between the β-strategies occurred (see above), with the consequenc that up to the occurrence of the switch the "effect" of β_7 is equivalent to that of β_5.

TABLE I. COMPUTATIONAL RESULTS USING THE BRANCH-AND-BOUND ALGORITHM.

PROBLEM CHARACTERISTICS							BRANCH-AND-BOUND SOLUTION ON IBM 370/158								OTHER ALGORITHMS			
Problem no. (reference)	No. of elements	Cycle time	Knowamin no of stations	No. of precedence relations	ILP-MODEL Thangavelu and Shetty: No. of constraints M+N+R+1	ILP-MODEL Thangavelu and Shetty: No. of 0-1 variables	No. of nodes generated	B-parameter	No of stations: Normal	No of stations: Parallel	CPU-times in seconds	Mean CPU-time	Variance CPU-time	Mean no. of nodes	Variance no. of nodes	No. of stations	CPU-time (seconds)	Method reference	Computer
1(5)	11	12	4	12	⩾28	⩾44	6	β_1	4	0	0.005	0.005	3×10^{-7}	6	0	-	-	-	-
2(8)	11	10	5	13	31	66	10	β_1	5	0	0.009	0.010	2×10^{-7}	10	0	5	0.853	(14)	UNIVAC 1108
		12	4	13	31	66	9	β_1	4	0	0.008	0.0083	2.67×10^{-7}	9	0	4	0.050	(14)	UNIVAC 1108
3(11)	19	179	6	30	⩾56	⩾114	36	β_3	6	0	0.054	0.0563	2.27×10^{-6}	36	0	6	-	(11)	IBM 7090
4(16)	21	14	8	27	59	210	21	β_1	8	0	0.024	0.0252	9.67×10^{-7}	21	0	8	0.078	(14)	UNIVAC 1108
		18	6	27	58	189	20	β_1	6	0	0.020	0.0217	2.67×10^{-7}	20	0	6	0.146	(14)	UNIVAC 1108
		19	6	27	⩾55	⩾126	12	β_1	6	0	0.015	0.0157	6.67×10^{-7}	12	0	-	-	-	-
		20	6	27	⩾57	168	19	β_1	6	0	0.018	0.01983	9.67×10^{-7}	19	0	6	0.070	(14)	UNIVAC 1108
		21	5	27	57	168	15	β_1	5	0	0.016	0.04067	0.00342	15	0	5	0.153	(14)	UNIVAC 1108
5(13)	42	396	?	57	⩾112	⩾504	148	β_3	12	0	0.358	0.92834	0.22580	240	5561	-	-	-	-
		399	?	57	⩾112	⩾504	99	β_7	12	0	0.168	0.17283	1.37667	99	0	-	-	-	-
		441	?	57	⩾111	⩾462	327	β_7	11	0	2.109	2.15983	0.00079	324	3	-	-	-	-
		474	?	57	⩾110	⩾420	149	β_4	10	0	0.345	0.373	0.00075	141	96	-	-	-	-
6(9)	45	69	8	62	118	450	440	β_2	8	0	1.757	2.33525	0.82306	464	3165	8	4.949	(14)	UNIVAC 1108
7(11)	48	161	15	66	⩾130	⩾720	107	β_2	15	0	0.199	0.21183	0.00024	107	0	15	57.6	(11)	IBM 7090

TABLE I (CONTINUED)

PROBLEM CHARACTERISTICS							BRANCH-AND-BOUND SOLUTION ON IBM 370/158									OTHER ALGORITHMS			
Problem no. (reference)	No. of elements	Cycle time	Known min no. of stations	No. of precedence relations	ILP-MODEL Thangavelu and Shetty: No. of constraints M+N+R+1	ILP-MODEL Thangavelu and Shetty: No. of 0-1 variables	No. of nodes generated	β-parameter	No. of stations: Normal	No. of stations: Parallel	CPU-times in seconds	Mean CPU-time	Variance CPU-time	Mean no. of nodes	Variance no. of nodes	No. of stations	CPU-time (seconds)	Method reference	Computer
8(16)	70	83	?	86	≥204	≥3290	2944	β_1	32	15	88.008	91.16433	11.6676	2.930	1361	47	-	(12)	-
		86	?	86	≥203	≥3220	2377	β_1	32	14	55.484	57.95683	1.8322	2.364	110	46	-	(12)	-
		89	?	86	≥200	≥3010	1421	β_2	31	12	18.159	18.751	0.1018	1.451	18879	43	-	(12)	-
		95	?	86	≥197	≥2800	2860	β_1	30	10	51.654	56.1405	14.4438	2.898	4166	40	-	(12)	-
		170	?	86	≥179	≥1540	2748	β_5	21	1	139.579	-	-	-	-	23	-	(12)	-
		173	22	86	≥179	≥1540	1413	β_6	21	1	37.168	49.0642	73.3490	1.644	15701	22	-	(12)	-
		176	22	86	≥179	≥1540	891	β_5	21	1	7.287	8.3747	1.5102	909	161	22	-	(12)	-
		179	21	86	≥178	≥1470	470	β_6	20	1	2.893	3.1758	0.3597	546	3085	21	-	(12)	-
		182	21	86	≥178	≥1470	222	β_7	20	1	.674	1.0715	0.0333	291	515	21	-	(12)	-
		346	11	86	≥168	≥770	1279	β_2	11	0	15.960	-	-	-	-	11	-	(12)	-
		349	11	86	≥168	≥770	618	β_5	11	0	10.540	-	-	-	-	11	-	(12)	-
		352	11	86	≥168	≥770	657	β_2	11	0	13.912	-	-	-	-	11	-	(12)	-
		355	11	86	≥168	≥770	2887	β_1	11	0	61.673	-	-	-	-	11	-	(12)	-
		358	11	86	≥168	≥770	881	β_2	11	0	21.603	-	-	-	-	11	-	(12)	-

Table II : Detailed branch-and-bound results for Tonge's 70-element problem.

Cycle time	β_1	β_2	β_3	β_4	β_5	β_6	β_7
83	[a] 88.008 [a] 2944	88.256 2902	89.678 2978	88.245 2914	93.772 2906	96.740 2977	90.295 2904
86	55.484 2377	57.134 2355	57.613 2372	56.186 2377	60.034 2353	58.909 2370	57.865 2355
89	19.120 1629	18.159 1421	18.261 1346	18.837 1625	18.537 1377	19.031 1348	18.720 1378
95	51.654 2860	52.996 2870	53.813 2851	54.789 2858	53.430 2870	61.978 3021	59.837 2916
170	177.776 3118	> 3 min	> 3 min	> 3 min	139.579 2748	> 3 min	> 3 min
173	41.091 1616	42.819 1606	47.886 1582	46.349 1551	51.360 1616	37.168 1413	64.880 1895
176	7.522 913	7.806 906	8.227 906	8.464 930	7.287 891	10.754 907	7.475 891
179	3.212 571	3.038 566	3.026 470	4.331 630	2.721 518	2.893 470	2.707 518
182	.878 253	.968 293	1.316 313	.935 279	1.061 295	1.271 313	.674 222
346	37.457 3026	15.960 1279	> 3 min	41.615 2820	>3 min	>3 min	>3 min
349	> 3.500	> 3.500	12.004 727	48.136 3040	10.540 618	12.005 727	10.574 618
352	> 3 min	13.912 657	> 3 min	> 3 min	16.281 817	> 3 min	16.325 817
355	61.673 2887	> 3 min	> 3 min	> 3.500	> 3.500	> 3 min	> 3.500
358	52.396 2177	21.603 881	> 3 min	> 3.500	24.756 1278	> 3 min	24.750 1278

[a] The first entry indicates CPU-time in seconds; the second entry gives the number of nodes generated.

CONCLUDING REMARKS.

The computational results discussed in the previous section indicate the branch-and-bound method for solving the single-model deterministic assembly line balancing problem to be more than competitive to previously reported optimal solution procedures. Computer times obtained reveal that problems up to 50 elements can be solved within 2 seconds of CPU-time, and that for the realistic 70-element problem set included in the computational experiment, CPU-time on the average amounts up to approximately half a minute.

The following factors seem to have contributed to make the algorithm rather efficient :

i) The way in which the information is stored and retrieved by our procedure seems to be very efficient.

ii) The overall set-up of the branching strategy - i.e., once all work elements assigned, the procedure may stop thus eliminating extensive backtracking - looks very promising.

iii) Bounding and dominance arguments together seem to be strong enough, although the lower bound standing alone seems too insensitive to yield a sufficient pruning effect in order to keep the number of nodes generated within acceptable limits.

iv) The β-parameters allow the procedure to run to completion for all problems that were tested. The behaviour of these parameters, however, is not much consistent and the net effect is rather negligible for small-sized problems. For larger problems certain β-parameters (especially β_5 and β_7) allow us to obtain an optimal solution within the limits of time and memory, which could otherwise not have been obtained.

REFERENCES

1. BUSSMANN, K.F. et al., "Ein Vergleich von Fliessbandabstimmungsverfahren", OR und Datenverarbeitung bei der Produktionsplanung (Bussmann, K.F., Mertens, P. (eds)), C.E. Poeschel Verlag, Stuttgart (1968), pp. 313-356.

2. CAULEY, J.M., "A Review of Assembly Line Balancing Algorithms", Proceedings of 19th Annual Conference, American Institute of Industrial Engineers, New York, 1968.

3. DAR-EL (MANSOOR), E.M., "Solving Large Single-Model Assembly Line Balancing Problems - A comparative study", AIIE Transactions, Vol. 7(1975), pp. 302-310.

4. DAVIS, E.W., HEIDORN, G.E., "An Algorithm for Optimal Project Scheduling under Multiple Resource Constraints", Management Science, Vol. 17(1971), pp. B-803 - B-816.

5. ELMAGHRABY, S.E., "Activity Networks", to appear

6. HELD, M., KARP, R.M., SHARESHIAN, R., "Assembly Line Balancing - Dynamic Programming with Precedence Constraints", Operations Research, Vol. 11(1963), pp. 442-459.

7. IGNALL, E., "A Review of Assembly Line Balancing", Journal of Industrial Engineering, Vol. 16(1965), pp. 244-254.

8. JACKSON, J.R., "A Computing Procedure for a Line Balancing Problem", Management Science, Vol. 2(1956), pp. 261-271.

9. KILBRIDGE, M.D., WESTER, L., "A Review of Analytical Systems of Line Balancing", Operations Research, Vol. 10(1962), pp. 626-638.

10. KOHLER, W.H., STEIGLITZ, K., "Enumerative and Iterative Computational Approaches", Chapter 6 in "Computers and Job-Shop Scheduling Theory" (COFFMAN, E.G. Jr. (ed.)), Wiley, 1976.

11. MOODIE, C.L., "A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times", unpublished Ph.D. Thesis, University of California, Los Angeles, 1964.

12. NEVINS, A.J., "Assembly Line Balancing Using Best Bud Search", Management Science, Vol. 18(1972), pp. 529-539.

13. REEVE, N., "Balancing Continuous Stochastic Assembly Lines", Ph.D. Thesis, State University of New York at Buffalo, 1971.

14. THANGAVELU, S.R., SHETTY, C.M., "Assembly Line Balancing by Zero-One Integer Programming", AIIE Transactions, Vol. III(1971), pp. 61-68.

15. TONGE, F.M., "Summary of the Heuristic Line Balancing Procedure", Management Science, Vol. 7 (1960), pp. 21-42.

16. TONGE, F.M., "A Heuristic Program for Assembly Line Balancing", Prentice-Hall, Inc., 1961.

17. TONGE, F.M., "Assembly Line Balancing Using Probabilistic Combinations of Heuristics", Management Science, Vol. 11(1965), pp. 727-735.